**源码学习**

[一、 Xen源码学习 3](#_Toc62458239)

[（一） pci直通相关 3](#_Toc62458240)

[1. 直通的基本介绍 3](#_Toc62458241)

[2. 直通的基本流程 4](#_Toc62458242)

[a） 将设备从Domain0解绑（unbind） 4](#_Toc62458243)

[b） 使设备处于待分配状态（assignable） 4](#_Toc62458244)

[c） 将设备分配给某一个DomainU（attach） 4](#_Toc62458245)

[d） 将设备从虚拟机移除（detach） 5](#_Toc62458246)

[e） 将该设备恢复为待分配状态（remove） 5](#_Toc62458247)

[f） 恢复Domain0对该设备的访问与使用权限（rebind） 5](#_Toc62458248)

[g） 通用函数功能与宏 5](#_Toc62458249)

[21-25 宏定义：（pci bdf的四种表达形式，以及msi中断传递和电源管理） 5](#_Toc62458250)

[27-37 函数 pcidev\_encode\_bdf 5](#_Toc62458251)

[39-48 函数 pcidev\_struct\_fill 5](#_Toc62458252)

[50-64 函数 libxl\_create\_pci\_backend\_device 6](#_Toc62458253)

[66-76 函数 libxl\_\_device\_from\_pcidev 6](#_Toc62458254)

[78-110 函数 libxl\_\_create\_pci\_backend 6](#_Toc62458255)

[<---一些注释---> 6](#_Toc62458256)

[112-190 函数 libxl\_\_device\_pci\_add\_xenstore 6](#_Toc62458257)

[192-306 函数 libxl\_\_device\_pci\_remove\_xenstore 7](#_Toc62458258)

[308-350 函数 get\_all\_assigned\_devices 7](#_Toc62458259)

[352-370 函数 is\_pcidev\_in\_array 7](#_Toc62458260)

[372-397 函数 sysfs\_write\_bdf 7](#_Toc62458261)

[399-446 函数 libxl\_device\_pci\_assignable\_list 7](#_Toc62458262)

[448-484 函数 sysfs\_dev\_unbind 8](#_Toc62458263)

[486-511 函数 sysfs\_dev\_get\_vendor 8](#_Toc62458264)

[513-538 函数 sysfs\_dev\_get\_device 8](#_Toc62458265)

[540-566 函数 sysfs\_dev\_get\_class 8](#_Toc62458266)

[568-595 函数 libxl\_\_is\_igd\_vga\_passthru 8](#_Toc62458267)

[597-638 函数 pciback\_dev\_has\_slot 9](#_Toc62458268)

[640-666 函数 pciback\_dev\_is\_assigned 9](#_Toc62458269)

[668-688 函数 pciback\_dev\_assign 9](#_Toc62458270)

[690-707 函数 pciback\_dev\_unassign 9](#_Toc62458271)

[709-709 宏定义 10](#_Toc62458272)

[711-725 函数 pci\_assignable\_driver\_path\_write 10](#_Toc62458273)

[727-737 函数 pci\_assignable\_driver\_path\_read 10](#_Toc62458274)

[739-751 函数 pci\_assignable\_driver\_path\_remove 10](#_Toc62458275)

[753-828 函数 libxl\_\_device\_pci\_assignable\_add 10](#_Toc62458276)

[830-879 函数 libxl\_\_device\_pci\_assignable\_remove 10](#_Toc62458277)

[881-891 函数 libxl\_device\_pci\_assignable\_add 10](#_Toc62458278)

[894-904 函数 libxl\_device\_pci\_assignable\_remove 11](#_Toc62458279)

[911-953 函数 pci\_multifunction\_check 11](#_Toc62458280)

[955-967 函数 pci\_ins\_check 11](#_Toc62458281)

[969-1008 函数 qemu\_pci\_add\_xenstore 11](#_Toc62458282)

[1010-1137 函数 do\_pci\_add 11](#_Toc62458283)

[1139-1174 函数 libxl\_\_device\_pci\_reset 12](#_Toc62458284)

[1176-1183 函数 libxl\_\_device\_pci\_setdefault 12](#_Toc62458285)

[1185-1194 函数 libxl\_device\_pci\_add 12](#_Toc62458286)

[1196-1211 函数 libxl\_pcidev\_assignable 12](#_Toc62458287)

[1213-1310 函数 libxl\_\_device\_pci\_add 12](#_Toc62458288)

[1312-1340 函数 libxl\_\_add\_pcidevs 13](#_Toc62458289)

[1342-1376 函数 qemu\_pci\_remove\_xenstore 13](#_Toc62458290)

[1378-1379 函数定义 libxl\_\_device\_pci\_remove\_common 13](#_Toc62458291)

[1381-1509 函数 do\_pci\_remove 13](#_Toc62458292)

[1511-1544 函数 libxl\_\_device\_pci\_remove\_common 14](#_Toc62458293)

[1546-1558 函数 libxl\_device\_pci\_remove 14](#_Toc62458294)

[1560-1571 函数 libxl\_device\_pci\_destroy 14](#_Toc62458295)

[1573-1609 函数 libxl\_\_device\_pci\_from\_xs\_be 14](#_Toc62458296)

[1611-1636 函数 libxl\_device\_pci\_list 14](#_Toc62458297)

[1638-1659 函数 libxl\_\_device\_pci\_destroy\_all 14](#_Toc62458298)

[1661-1704 函数 libxl\_\_grant\_vga\_iomem\_permission 15](#_Toc62458299)

[1706-1710 函数 libxl\_device\_pci\_compare 15](#_Toc62458300)

[1712-1712 宏定义 15](#_Toc62458301)

[1714-1714 函数声明 15](#_Toc62458302)

[h） 其他文件中相关函数与宏 15](#_Toc62458303)

[<--- 头文件内---> 15](#_Toc62458304)

[<---关于 libxl\_device\_pci 结构体---> 16](#_Toc62458305)

[<---关于 libxl\_\_gc 结构体---> 16](#_Toc62458306)

[<---关于 flexarray\_t 结构体---> 16](#_Toc62458307)

[<---关于 xenbus\_state （XenbusState）枚举类型---> 17](#_Toc62458308)

[<---关于 libxl\_device 结构体---> 17](#_Toc62458309)

[<---关于 libxl\_\_device\_kind 枚举类型---> 17](#_Toc62458310)

[libxl\_\_xs\_read：（tools/libxl/libxl\_xshelp.c） 18](#_Toc62458311)

[libxl\_\_xs\_writev：（tools/libxl/libxl\_xshelp.c） 18](#_Toc62458312)

[atoi：（<stdlib.h>） 18](#_Toc62458313)

[device\_add\_domain\_config：（tools/libxl/libxl\_device.c） 18](#_Toc62458314)

[<---关于 libxl\_domain\_config 结构体---> 18](#_Toc62458315)

[<---关于 libxl\_device\_type 结构体---> 19](#_Toc62458316)

[<---关于 libxl\_domain\_type 枚举类型---> 21](#_Toc62458317)

[<---关于 libxl\_asyncop\_how 结构体---> 21](#_Toc62458318)

[<---关于 libxl\_\_ao 结构体---> 22](#_Toc62458319)

[附件B：tools/libxl/flexarray相关 23](#_Toc62458320)

[1. flexarray.h 共55行 23](#_Toc62458321)

# Xen源码学习

源码版本：xen 4.11.4

## 名词解释

##### Xen

Xen是一个开放源代码虚拟机监视器，由剑桥大学开发。

##### Hypervisor

虚拟机监控器（VMM），也叫Hypervisor。Hypervisor层在硬件与虚拟机之间，是必须最先载入到系统环境的第一层。

##### Domain0

[Xen](#_Xen)中具有最高权限的虚拟机。由于[Hypervisor](#_Hypervisor)中不包含任何与硬件对话的驱动，也没有与管理员对话的接口，因此这些驱动就要由Domain0来提供。

通过Domain0，管理员可以利用一些Xen工具来创建其它虚拟机（称为[DomainU](#_DomainU)）。这些DomainU也叫无特权Domain，这是因为在基于i386的CPU架构中，它们绝不会享有最高优先级，只有Domain0才可以。

在Domain0中，还会载入一个xend进程。这个进程会管理所有其它虚拟机，并提供这些虚拟机控制台的访问。在创建虚拟机时，管理员使用配置程序与Domain0直接对话。

下文有时会简写为“DOM 0”。

##### DomainU

又称为“无特权Domain”，只拥有受限的权限，默认情况下完全被[Domain0](#_Domain0)控制。

下文有时会简写为“DOM U”。

##### X86

X86架构（The X86 architecture）是微处理器执行的计算机语言指令集，指一个intel通用计算机系列的标准编号缩写，也标识一套通用的计算机指令集合。

##### PCI

PCI是Peripheral Component Interconnect（外设部件互连标准）的缩写，它是目前个人电脑中使用最为广泛的接口。

##### QEMU

QEMU是一套模拟处理器，在GNU/Linux平台上使用广泛。

## pci直通相关

### 直通的基本介绍

在Xen虚拟化环境下，虚拟机到硬件之间总共有三层结构，可以按照下图1.1参考理解：

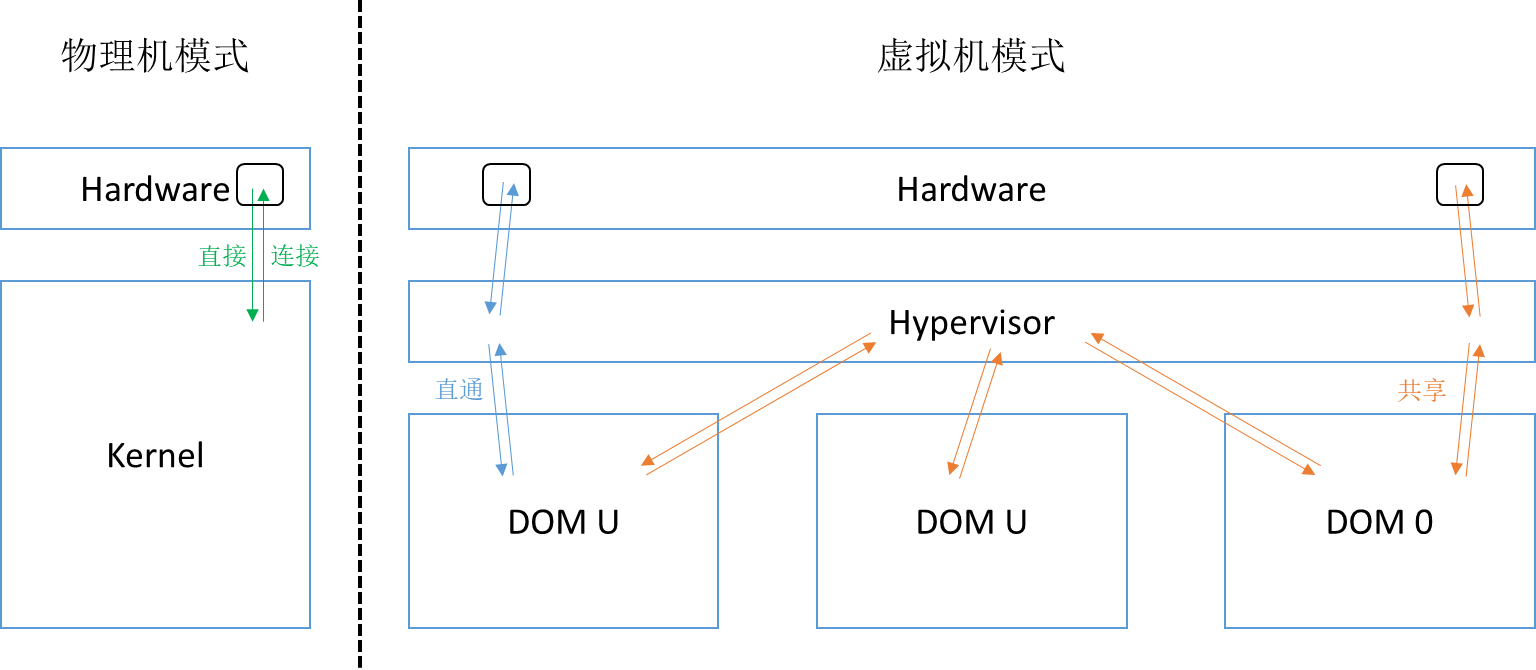


图1.1 Xen的设备访问流程

虚拟机在[Hypervisor](#_Hypervisor)层级会有一部分模块专门用于处理“设备”的配置与使用请求，我们可以将之称为“设备模型”；在[X86](#_X86)架构下，这些设备通常是以[pci](#_PCI)设备的模式挂载到总线上，因此在X86的虚拟化环境中，对pci设备（或设备模型）的处理就尤为重要。

处于安全性的考虑，Xen的Hypervisor将旗下的虚拟机划分为两大类：默认具有全部设备控制权限的[Domain0](#_Domain0)，以及默认只能访问虚拟出的硬件环境的[DomainU](#_DomainU)；在更为早古的Xen环境下，还有一类stubdom域专门用于将设备与虚拟域隔离，但是在将相关操作交由[qemu](#_QEMU)处理后，这一种域类型似乎被摒弃了。

如上图所示，默认情况下DomainU与真实的物理硬件会被Hypervisor隔离，并将相关的配置、访问请求转交给Domain0进行处理。这样的处理方式，其优点在于可以确保硬件的安全性，并且同一个硬件可以通过某些手段同时分配给多个不同的DomainU进行使用；但是缺点也很明显，所有对于物理硬件的访问都会经过一个比较长（或者复杂）的流程，会造成时间以及算力的浪费。我们可以将这种对物理硬件的管理方式称为“共享”模式，其使用流程可以参考上图1.1中的橙色部分内容。

为解决上述模式的不足，虚拟化软件还提供了另一种虚拟机（DomainU）访问物理硬件的模式：允许Domain0授予某一个DomainU对于某一个硬件的直接访问权限，在此之后当前虚拟环境下将有且仅有这一个DomainU可以访问并使用该物理硬件（Domain0也会失去该硬件的访问权限）。这意味着Domain0对于该DomainU使用这个物理硬件有着充分的“信任”，这种状态下的优劣与之前正好相反：在限制了只有这一个DomainU可以使用该硬件的同时，却可以充分确保使用物理硬件的时间与算力成本的最优化。因此，在某些应用场景下，使用这一种“直通（PassThrough）”模式会更符合实际的需求。

在Xen虚拟化环境下，有两种方式可以配置设备的直通：将配置需求添加到建立虚机的配置文件中，或者在虚机启动后通过xl命令动态的添加需要直通的设备。这两种方式最终都会调用到tools/libxl/libxl\_pci.c中的相关函数，因此本文决定对该函数进行分析与理解，以便于理清相关流程中的各处利害关系。

### 直通的基本流程

在x86架构下的Xen虚拟化环境下，一个pci设备想要分配到一个DomainU总共需要以下几步：将设备从Domain0解绑（unbind）--->使其处于待分配状态（assignable）--->将设备分配给某一个DomainU（attach）--->在DomainU停止运行后或其他必要时间，将设备从虚拟机移除（detach）--->将该设备恢复为待分配状态（remove）--->如果需要的话，恢复Domain0对该设备的访问与使用权限（rebind）。下文将结合源码中的相关流程梳理这些步骤的实现流程。

#### 将设备从Domain0解绑（unbind）

该流程的主函数在[448-484行](#_448-484_函数_sysfs_dev_unbind)，会被[assignable](#_使设备处于待分配状态（assignable）)操作调用。

x86架构下的Linux系统会在/sys/bus/pci路径下存放着devices和drivers的信息，这些路径下会有一系列的link来将设备和其对应的驱动匹配到一起（或者说，“绑定（bind）”到一起），只有在合适的链接匹配下设备才可以正常运行、工作。

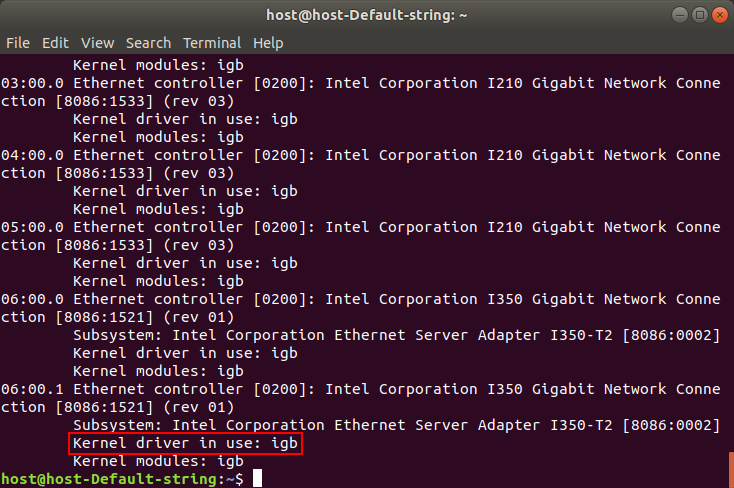
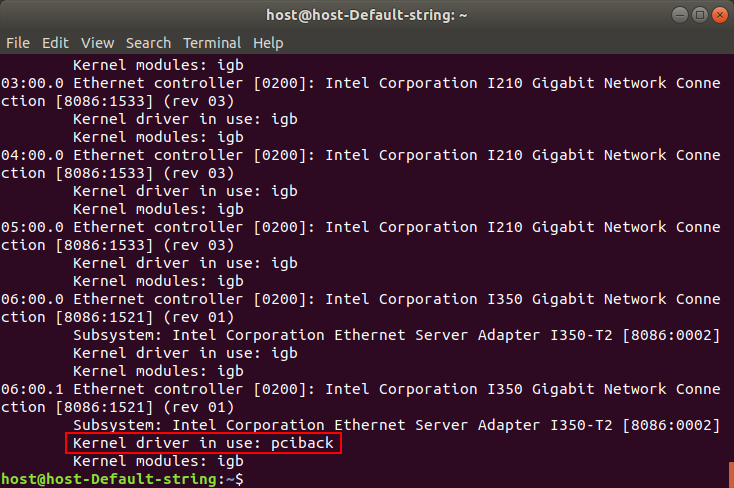
 

图1.2 左：未分配的设备；右：做好直通准备的设备

如图1.2所示，针对Xen虚拟化环境，其设计了一类名为pciback的驱动，所有应做好直通准备的设备都应该被挂载到这个驱动上（使用lspci –nnk命令可以查看当前所有pci设备挂载的驱动）；如果没有将该设备设定为“开机自动分配”状态（需要使用xen-pciback.hide=(BB:DD.F)以及xen-pciback.passthrough=1这两个GRUB\_CMDLINE\_LINUX\_DEFAULT参数以实现开机自动分配），那么就需要手动进行设备的分配操作。在此之前，由于在Linux内核启动过程中已经将设备分配至其默认的驱动上，因此若要将其设置为等待直通分配的状态，则应该先将这个设备从原先的驱动上解绑（[unbind](#_将设备从Domain0解绑（unbind）)），并且如果需要的话，还得备份一下这个设备原先使用的驱动的路径以便于日后将该设备重新绑定（[rebind](#_恢复Domain0对该设备的访问与使用权限（rebind）)）至原生的驱动。

#### 使设备处于待分配状态（assignable）

该流程的主函数在[881-891行](#_881-891_函数_libxl_device_pci_assigna)，会被xl\_pci.c中的相关函数调用，并在执行xl pci-assignable-add命令时启动；或根据虚机cfg文件的相关设置，在[attach](#_将设备分配给某一个DomainU（attach）)的流程中自动检测并直接调用[753-828行](#_753-828_函数_libxl__device_pci_assign)的函数进行注册。

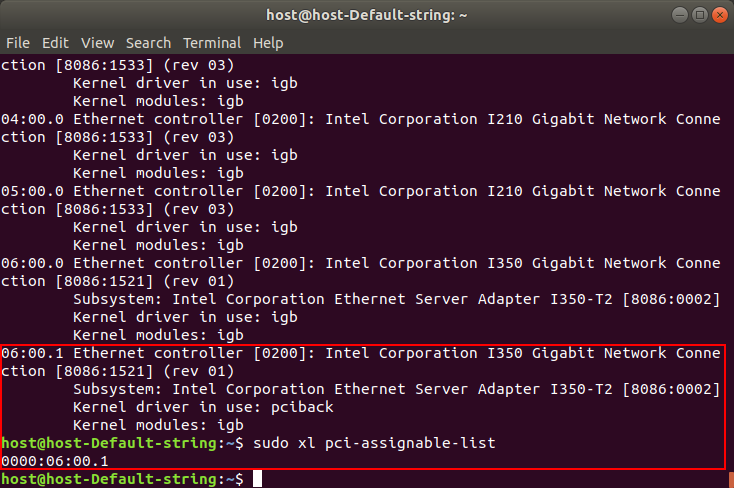


图1.3 查看所有处于assignable状态的设备（BDF列表）

如上文所述，只有被绑定到pciback驱动上的pci设备才可以被成功分配给下位的虚拟机（分配后该设备便处于[assignable](#_使设备处于待分配状态（assignable）)状态，可以使用sudo xl pci-assignable-list命令（在[399-446行](#_399-446_函数_libxl_device_pci_assigna)实现，会被xl\_pci.c中的相关函数调用），如图1.3所示）；因此该函数的主要操作就是调用执行[unbind](#_将设备从Domain0解绑（unbind）)操作并将解绑后的pci设备绑定到pciback“驱动”上。在该操作正常完成后，这个pci设备就可以随时准备着分配给虚拟机了。

所有经历过[assignable](#_使设备处于待分配状态（assignable）)流程的设备，在被[remove](#_将该设备恢复为待分配状态（remove）)之前其pci状态都不会发生改变，即在Domain0下Linux的用户态层级是看不出来任何区别的，只会在Xen中进行相应的标记与区分。

#### 将设备分配给某一个DomainU（attach）

该流程的主函数在[1185-1194行](#_1185-1194_函数_libxl_device_pci_add)，会被xl\_pci.c中的相关函数调用，并在执行xl pci-attach命令时启动；若该pci设备被配置在了虚机的cfg文件中，则会由[1312-1340行](#_1312-1340_函数_libxl__add_pcidevs)的函数进行批量处理（启动过程中存在一些回调关系，虚机建立起始点可以参考tools/xl/xl\_vmcontrol.c的641行）。

为了加快处理速度，该操作还会设计一些异步运行的[ao](#_<---关于_libxl__ao_结构体--->)操作（由[ao\_how](#_<---关于_libxl_asyncop_how_结构体--->)进行配置、处理），相关流程可以参考tools/libxl/libxl\_event.c。

[](https://wiki.xenproject.org/wiki/QEMU_vs_qemu-traditionnal_Feature_Comparison)

图1.4 两种device\_model模式的功能对比

（截图时的wiki链接版本更新时间：15:14, 29 October 2014）

在这个流程会涉及到两种将pci设备添加到虚机（这是[attach](#_将设备分配给某一个DomainU（attach）)操作的最终目的）的实现方式：qemu-xen与qemu-xen-traditional，其功能对比如图1.4所述。后者是内置在Xen源码内部的一系列操作的实现，交给xs（xenstore）去进行设备的管理、注册、挂载等操作，这个版本的功能实现的比较稳定，但是已经不再更新了，如果日后有新的需求可能会遇到问题；前者是基于qemu的一系列操作的实现，有些功能会在实现过程中产生稳定性的问题（VGA在这一模式下暂未实现直通，可以参考[这个链接](https://wiki.xenproject.org/wiki/QEMU_Upstream#Missing_feature_from_the_good_old_qemu-dm)中的描述（编写该文档时的wiki链接版本更新时间：10:09, 14 February 2019））。根据目前的测试结果，在Xen的4.11.4版本下，后者可以实现USB、VGA和非系统盘的SATA的直通而前者不能；但是在更新到4.14.1版本后（4.12应该也是），后者可以实现USB、VGA和非系统盘的SATA的直通，前者尽管仍旧不能稳定直通前两点，但是可以实现SATA作为系统盘以及非系统盘的直通。

针对两种模式的说明和配置方式可以参考[device\_model\_version](#_device_model_version="DEVICE-MODEL" )中的介绍。

在进行了[attach](#_将设备分配给某一个DomainU（attach）)流程之后，该pci设备会从pci-assignable-list中移除，以防止将同一个pci设备同时分配给多个不同的虚机。

#### 将设备从虚拟机移除（detach）

该流程的主函数在[1560-1571行](#_1560-1571_函数_libxl_device_pci_destr)，会被xl\_pci.c中的相关函数调用，并在执行xl pci-detach命令时启动；当添加-f参数时，会使能force标志并从[1546-1558行](#_1546-1558_函数_libxl_device_pci_remov)开始执行；当虚机被关闭时（libxl\_domain.c的1063行），则会从[1638-1659行](#_1638-1659_函数_libxl__device_pci_dest)进入，遍历并移除所有分配出的pci设备。上述三种方式的具体实现均在[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)中。

该操作可以解除虚机（此处特指[DomainU](#_DomainU)）对于设备的“占用”状态，相关“占用”标记会存储在xs中。流程结束后，设备会恢复到pci-assignable-list中以便于再次分配。

#### 将该设备恢复为待分配状态（remove）

该流程的主函数在[894-904行](#_894-904_函数_libxl_device_pci_assigna)，会被xl\_pci.c中的相关函数调用，并在执行xl pci-assignable-remove命令时启动，-r参数会同时使能[rebind](#_恢复Domain0对该设备的访问与使用权限（rebind）)功能。

该操作可以解除[Xen](#_Xen)对于设备的“占用”状态，相关“占用”状态标记在pciback驱动中进行组织管理。

#### 恢复Domain0对该设备的访问与使用权限（rebind）

该流程的主函数在[858-876行](#_894-904_函数_libxl_device_pci_assigna)，是[remove](#_将该设备恢复为待分配状态（remove）)操作的一个分支。由于其在程序上的独立性，将其单独列了出来。

该操作可以恢复pci设备原先的驱动，在此之后RootOS（即[Domain0](#_Domain0)）就可以像往常一样使用这个设备了。

#### 通用函数功能与宏

##### 21-25 宏定义：（pci bdf的四种表达形式，以及msi中断传递和电源管理）

#define PCI\_BDF "%04x:%02x:%02x.%01x"

#define PCI\_BDF\_SHORT "%02x:%02x.%01x"

#define PCI\_BDF\_VDEVFN "%04x:%02x:%02x.%01x@%02x"

#define PCI\_OPTIONS "msitranslate=%d,power\_mgmt=%d"

#define PCI\_BDF\_XSPATH "%04x-%02x-%02x-%01x"

##### 27-37 函数 pcidev\_encode\_bdf

输入：libxl\_device\_pci \*pcidev

输出：unsigned int

描述：根据输入的pci设备类型返回其bdf。

##### 39-48 函数 pcidev\_struct\_fill

输入：libxl\_device\_pci \*pcidev, unsigned int domain, unsigned int bus, unsigned int dev, unsigned int func, unsigned int vdevfn

输出：void

描述：填充 libxl\_device\_pci 结构体BDF相关数据值。

##### 50-64 函数 libxl\_create\_pci\_backend\_device

输入：libxl\_\_gc \*gc, flexarray\_t \*back, int num, libxl\_device\_pci \*pcidev

输出：void

描述：将一系列pci设备信息（例如BDF等）以字符串组形式添加进back中。

##### 66-76 函数 libxl\_\_device\_from\_pcidev

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, libxl\_\_device \*device

输出：void

描述：将pci设备定义为标准设备类型，并标注为PCI。

##### 78-110 函数 libxl\_\_create\_pci\_backend

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int num

输出：int

描述：建立用于包含前后端设备信息的flexarray，然后将pci设备转化为标准设备类型，将相关信息填入到flexarray中（向back中填入简短的前端信息与所有pci设备的信息以及设备数；向front中填入简短的后端信息），最后将标准设备模型与前后端array->data中的所有字符串信息提交到新建的xs事务中并返回结果状态（权限相关、进程信号集）。

备注：libxl\_\_device\_generic\_add函数相关内容牵扯程序关联较深，暂未深入分析，可能存在一定误解。可以关注libxl\_device.c、xs.c中的相关内容，总计2118行、1466行。

##### <---一些注释--->

--libxl\_device.c：203-207

----控制台0是一个特殊的情形。它不会使用通常意义上的PV状态机；但是前端目录历来包含其他信息（例如如VNC端口），这部分是我们不希望guest修改的。

--libxl\_device.c：236-258

----我们还为libxl路径中后端的所有内容制作了副本。这意味着我们不需要信任后端。在理想状态下，这些信息是不被需要的；我们将使用来自json配置中的信息来代替。但是libxl中仍然有一些地方试图从xenstore中重建配置。

----没有PV后端的设备（例如，通过QEMU模拟的USB设备）只被写入了libxl路径。

----这种重复通常会产生重复的关键字，这将过时，但这是无所谓的，因为没人会读取这里的东西。例如，在/libxl/$guest/device/$kind/$devid/state路径下通常会开始包含一些XenbusStateInitialising ("1")之类的关键字，这正如在在/local/domain/$driverdom/backend/$guest/$kind/$devid/state路径下的副本一样，但是后者从来不会被更新。

----这种重复是多余和混乱的，但正如之前所讨论的，适当的修复比我们现在想做的更具挑战。

<---注释结束--->

##### 112-190 函数 libxl\_\_device\_pci\_add\_xenstore

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int starting

输出：int

描述：将pci设备信息添加进xenstore中（进行管理）（包含全局config配置）。新建并初始化config结构体、pci设备模型，将输入的pci设备模型复制，然后识别输入的domid下的后端路径以及其下的num\_devs，并将pci设备的相关信息添加进back中；然后将pci设备转化为标准设备类型并配置lock，根据domid获取d\_config数据，将之前复制出的pci设备模型信息添加到局部d\_config变量中；最后检查qemu的开启状态，激活xenstore事务（transaction），将更新后的局部d\_config变量更新至全局变量，向xs（之前获取到的）后端路径传递后端相关参数，验证传输状态（有失败重传），挂起xenstore事务（transaction）并结束。中间的一些临时变量会有dispose处理。

备注：含有较多涉及到xenstore的内容，该方向应为后续关注重点。

##### 192-306 函数 libxl\_\_device\_pci\_remove\_xenstore

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev

输出：int

描述：移除xenstore中的一个pci设备。根据输入的domid参数获得该虚拟机上pci设备的数量、虚机类型，分别读取每一个pci设备的BDF，直到找到输入的pci设备，然后恢复xs事务，将其中对应的pci设备状态置为“XenbusStateClosing”并重置xen总线状态，尝试结束xs事务（阶段性结束），如果可以正常结束（表示上述两部操作运行正常？）则继续，否则重试；然后再恢复xs事务，清除（不确定xs\_rm功能，根据字面理解）该pci设备在后端路径的信息，并将后续的pci设备编号依次前提，尝试结束xs事务（阶段性结束），如果可以正常结束则继续，否则重试；若删除前只剩一个设备，则销毁该设备（清理xs中该设备下的所有路径/信息）。在整个过程中，需要注意PV模式下对应虚机的后端的处理。

备注：含有较多涉及到xenstore的内容，该方向应为后续关注重点。

##### 308-350 函数 get\_all\_assigned\_devices

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*\*list, int \*num

输出：int

描述：搜寻在gc管理下的所有虚机下的所有pci设备的bdf，并依次填入list中。从全局gc中获取nd（不太清楚这个数是怎么来的，最终结果就是计算了一个字符串的长度，这个字符串是由xs的一个会话句柄返回的，该句柄为当前xs事务传输了一个XS\_DIRECTORY信号。）以及domlist字符串组，然后从每一个domlist中获得其下包含的pci设备数，然后将其中每一个设备的BDF的值依次填入list中对应结构体的对应位置。

##### 352-370 函数 is\_pcidev\_in\_array

输入：libxl\_device\_pci \*assigned, int num\_assigned, int dom, int bus, int dev, int func

输出：int

描述：根据BDF信息查找设备列表中是否存在对应的设备，是则返回1，否则返回0。

##### 372-397 函数 sysfs\_write\_bdf

输入：libxl\_\_gc \*gc, const char \* sysfs\_path, libxl\_device\_pci \*pcidev

输出：int

描述：向输入的sysfs路径中写入标准BDF信息。

##### 399-446 函数 libxl\_device\_pci\_assignable\_list

输入：libxl\_ctx \*ctx, int \*num

输出：libxl\_device\_pci \*

描述：获取已准备好直通但尚未被使用的pci设备列表。使用GC初始化ctx，在全局gc中搜寻所有已注册的pci设备信息（[308-350行](#_308-350_函数_get_all_assigned_devices)），然后打开pciback驱动的文件夹路径（存放着准备好进行分配的pci设备），并依次打开其中的每一个子目录，排出其中不符合BDF格式的路径或已经被注册的pci设备（通过[352-370行](#_352-370_函数_is_pcidev_in_array)进行判断），最后将剩下的pci设备依次填入设备列表（临时变量new对pcidevs申请新的内存空间扩容、pcidevs获取new申请的新内存空间、new指向其中刚刚扩容的空白部分、将new中的内容初始化清零、将检测到的符合要求的pci设备信息填入new（[39-48行](#_39-48_函数_pcidev_struct_fill)）、num自增）并return返回该列表。

备注：

----readdir();-->获取输入目录下的下一个入口指针。

----memset();-->清空指针指向内容的数据。

##### 448-484 函数 sysfs\_dev\_unbind

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, char \*\*driver\_path

输出：int

描述：如果存在输入的pci设备，则从其当前驱动程序中解绑设备；如果driver\_path是非空的，则将原始的驱动程序路径存储到这一变量中。根据输入的pci设备的信息，确认其对应的文件系统路径下的驱动路径（root OS中/sys/bus/pci/devices/"PCI\_BDF"/driver是一个link节点），并检查该目录是否存在（表示是否成功挂载着驱动）；若存在，则向该驱动的文件节点（经过了相对路径到绝对路径的转换）下的“unbind”词条写入该pci设备的BDF信息（使该驱动取消挂载该设备）。若输入的driver\_path非空，则向\*driver\_path写入之前挂载的驱动的绝对路径。

备注：

----lstat();-->用来获取linux操作系统下输入路径的文件的属性。

----realpath();-->将相对路径转换为绝对路径。

##### 486-511 函数 sysfs\_dev\_get\_vendor

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：uint16\_t

描述：从pci设备的文件系统路径中获取其供应商的信息。（root OS中/sys/bus/pci/devices/"PCI\_BDF"/vender存储了该设备的供应商信息）

##### 513-538 函数 sysfs\_dev\_get\_device

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：uint16\_t

描述：从pci设备的文件系统路径中获取其设备编号的信息。（root OS中/sys/bus/pci/devices/"PCI\_BDF"/device存储了该设备的设备编号信息）

##### 540-566 函数 sysfs\_dev\_get\_class

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, unsigned long \*class

输出：int

描述：从pci设备的文件系统路径中获取其类代码的信息。（root OS中/sys/bus/pci/devices/"PCI\_BDF"/class存储了该设备的类代码信息）

##### 568-595 函数 libxl\_\_is\_igd\_vga\_passthru

输入：libxl\_\_gc \*gc, const libxl\_domain\_config \*d\_config

输出：bool

描述：判断d\_config中是否存在vga的pci设备。遍历d\_config中的所有pci设备，先判断其供应商和设备号是否为杂项（0xFFFF），然后检查供应商是否为0x8086（代表英特尔），最后判断是否有类代码且类代码为0x030000（代表显示控制器），若存在满足上述所有条件的pci设备，则返回真；否则返回假。

备注：该项与VGA相关，可以查看其引用节点（libxl\_dm.c：733）

注释：

----一些设备可能需要一些调试才能正常工作。例如这里的IGD，我们必须传递一些特别的选项参数传递给qemu。

##### 597-638 函数 pciback\_dev\_has\_slot

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：int

描述：通过扫描/sys/bus/pci/drivers/pciback/slots路径以寻找pci设备的BDF。若找到输入的pci设备对应的BDF则返回1，否则返回0。

注释：

----关于slot的简短评论。我不知道slot是做什么的；然而，我已经通过实验确定了：

---- \* 在设备可以绑定到pciback之前，其BDF必须首先在pciback/slot中列出

---- \* 列出BDF的方法是将BDF写入pciback/new\_slot路径

---- \* 将相同的BDF写入pciback/new\_slot不是幂等的（安全的？）；它将在pciback/slot中导致两个（相同的）BDF条目

----目前尚不清楚在pciback/slot中存在两个（相同的）条目是否是一个问题。为了安全起见，这个代码做了保守的事情，它首先检查是否有一个slot，只有当其不存在时才添加一个新的。

##### 640-666 函数 pciback\_dev\_is\_assigned

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：int

描述：检查pci设备是否做好直通准备。验证先检查/sys/bus/pci/drivers/pciback路径是否存在，然后验证输入的pci设备对应的/sys/bus/pci/drivers/pciback/"PCI\_BDF"路径状态，若其状态正常（路径存在，pci设备已准备好）则返回1；若存在错误，则根据情况返回0（路径不存在）或-1。

备注：

----access();-->access函数用来判断指定的文件或目录是否存在(F\_OK)，已存在的文件或目录是否有可读(R\_OK)、可写(W\_OK)、可执行(X\_OK)权限。F\_OK、R\_OK、W\_OK、X\_OK这四种方式通过access函数中的第二个参数指定。如果指定的方式有效，则此函数返回0，否则返回-1。

##### 668-688 函数 pciback\_dev\_assign

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：int

描述：为pciback添加pci设备。先检查/sys/bus/pci/drivers/pciback/slot文件中是否存在输入的pci设备，若不存在则向/sys/bus/pci/drivers/pciback/new\_slot写入输入的pci设备的BDF信息；然后向/sys/bus/pci/drivers/pciback/bind中写入设备的BDF以使用pciback绑定设备。

##### 690-707 函数 pciback\_dev\_unassign

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：int

描述：从pciback移除pci设备。先解绑输入的pci设备（当前应绑定在pciback中），然后检查/sys/bus/pci/drivers/pciback/slot文件中是否存在输入的pci设备，若存在则向/sys/bus/pci/drivers/pciback/remove\_slot写入输入的pci设备的BDF信息。

##### 709-709 宏定义

#define PCIBACK\_INFO\_PATH "/libxl/pciback"

##### 711-725 函数 pci\_assignable\_driver\_path\_write

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, char \*driver\_path

输出：void

描述：向xs的/libxl/pciback/"PCI\_BDF\_XSPATH"/driver\_path中写入输入的driver\_path（原始驱动路径，见448-484）内容。

##### 727-737 函数 pci\_assignable\_driver\_path\_read

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev

输出：char \*

描述：从xs的/libxl/pciback/"PCI\_BDF\_XSPATH"/driver\_path中读取信息并返回。

##### 739-751 函数 pci\_assignable\_driver\_path\_remove

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev)

输出：void

描述：移除与输入的pci设备对应的xs中/libxl/pciback/"PCI\_BDF\_XSPATH"路径下的内容。

##### 753-828 函数 libxl\_\_device\_pci\_assignable\_add

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, int rebind

输出：int

描述：执行pci设备的解绑、并做好直通准备。先从输入的pci设备中备份一份本地的BDF信息变量，然后确认/sys/bus/pci/devices/"PCI\_BDF"路径是否存在（设备是否存在），并验证该设备是否已完成直通准备（关联[640-666行](#_640-666_函数_pciback_dev_is_assigned)，若已assignable则跳转到quarantine隔离区），然后尝试将其从原先的驱动程序中解绑（[448-484行](#_448-484_函数_sysfs_dev_unbind)，unbind相关操作），并根据输入参数确认其以后是否需要重新绑定（若需要，则将原先的驱动路径存储起来或输出已存储的备份信息，相关读写操作分别在[727-737行](#_727-737_函数_pci_assignable_driver_pa)与[711-725行](#_711-725_函数_pci_assignable_driver_pa)；若不需要，则清除对应设备在xs中的内容，关联[739-751行](#_739-751_函数_pci_assignable_driver_pa)），最后为pciback“驱动”添加该设备（关联[668-688行](#_668-688_函数_pciback_dev_assign)）。隔离区：将该pci设备注册进DOMID\_IO域（多级虚拟机的分配？），然后结束。

注释：DOMID\_IO只是一个哨兵域，没有任何实际的映射，所以总是应该传递XEN\_DOMCTL\_DEV\_RDM\_RELAXED以避免不必要的拒绝分配。

##### 830-879 函数 libxl\_\_device\_pci\_assignable\_remove

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, int rebind

输出：int

描述：先将该设备从DOMID\_IO域中注销，然后检查该设备是否为待分配的状态且已被pciback绑定（[640-666行](#_640-666_函数_pciback_dev_is_assigned)），若均满足则将该设备从pciback解绑（解绑后即代表着remove完成，在[690-707行](#_690-707_函数_pciback_dev_unassign)实现）；最后根据是否有其原生驱动路径的备份（通过[727-737行](#_727-737_函数_pci_assignable_driver_pa)检测）以及输入参数rebind的选择，决定是否将其绑定至原先的驱动中并清除对应设备在xs中备份的内容（通过[739-751行](#_739-751_函数_pci_assignable_driver_pa)清除备份）。

##### 881-891 函数 libxl\_device\_pci\_assignable\_add

输入：libxl\_ctx \*ctx, libxl\_device\_pci \*pcidev, int rebind

输出：int

描述：上述[753-828行](#_753-828_函数_libxl__device_pci_assign)的封装，额外包含了一些gc内容。

##### 894-904 函数 libxl\_device\_pci\_assignable\_remove

输入：libxl\_ctx \*ctx, libxl\_device\_pci \*pcidev, int rebind

输出：int

描述：上述[830-879行](#_830-879_函数_libxl__device_pci_assign)的封装，额外包含了一些gc内容。

##### 911-953 函数 pci\_multifunction\_check

输入：libxl\_\_gc \*gc, libxl\_device\_pci \*pcidev, unsigned int \*func\_mask

输出：int

描述：此函数检查pci设备的所有功能是否均绑定到pciback驱动程序（做好直通准备））。它还初始化了一个位掩码，用于记录存在于该设备上的函数号。检查/sys/bus/pci/devices路径下的所有子节点，验证其中与输入pci设备dom、bus、dev三个参数相同的所有节点，若其未被pciback绑定，则会提示错误信息；若其被pciback挂载，则将func\_mask的对应位置1。均无误返回0，否则返回1。

##### 955-967 函数 pci\_ins\_check

输入：libxl\_\_gc \*gc, uint32\_t domid, const char \*state, void \*priv

输出：int

描述：对输入的state字符串进行匹配，若为pci-insert-failed返回-1；若为pci-inserted返回0；若与输入的priv指向的字符串相同返回1；若均不同也返回1。

##### 969-1008 函数 qemu\_pci\_add\_xenstore

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev

输出：int

描述：添加pci设备，传输libxl\_device\_pci的信息（可能关联到qemu）。从ctx中获取当前的domid并记录该domain在xs中的state值，然后向xs的parameter路径写入pci设备的信息（包含BDF与中断、电源管理，若pci设备被指定了虚机中的DF参数，则需要同时传递这些参数），再向qemu传递“pci-ins”命令并等待设备模型deprecated（弃用）（检查state字符串，若状态变为pci-inserted则正常，否则报错），最后验证虚机中的DF参数是否配置正确，并向xs写入最终的运行状态。

备注：含有较多涉及到xenstore及qemu（可能）的内容，该方向应为后续关注重点。

----libxl\_\_wait\_for\_device\_model\_deprecated();--->若提供后两个参数（使用pci\_ins\_check函数），则当系统状态返回值为为pci-inserted时返回0，与第六参数相同时返回1（参考955-967）；若提供第三个参数，则若系统状态返回值与该参数相同时返回0。

##### 1010-1137 函数 do\_pci\_add

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int starting

输出：int

描述：完成pci设备的添加流程。先确认虚机是否为哨兵域（stubdom），然后针对hvm模式，会根据虚机使用的设备驱动模式的不同（传统与upstream）使用不同的方式注册pci设备（传统的直通在当前文件中[969-1008行](#_969-1008_函数_qemu_pci_add_xenstore)执行，upstream模式交给libxl\_qmp.c文件执行，那一处可能涉及到qemu的相关接口调用），接着打开当前pci设备的/sys/bus/pci/devices/"PCI\_BDF"/resource路径，循环7次扫描其中的内容，识别出其下所有I/O资源的start、end、flags，然后根据flag的最后一位判断是否为IO Bar（否则应为Memory Bar），来对xc的ioport进行许可申请（或对xc的iomem进行许可申请），最后读取当前pci设备的/sys/bus/pci/devices/"PCI\_BDF"/irq路径下的内容，若irq（中断）存在且不为零，则为这些中断建立物理设备的映射并对xc的irq进行许可申请，再将读写许可限制信息（permissive参数）导入，并进入结束流程。结束流程：如果是stubdom，则向xc注册设备；在这之后，如果有starting标志（使用xl命令时为0，从config中启动时为1），则向xs添加设备（[112-190行](#_112-190_函数_libxl__device_pci_add_xe)）。

##### 1139-1174 函数 libxl\_\_device\_pci\_reset

输入：libxl\_\_gc \*gc, unsigned int domain, unsigned int bus, unsigned int dev, unsigned int func

输出：int

描述：对pci设备进行reset操作。先尝试向/sys/bus/pci/drivers/pciback/do\_flr路径写入该设备的BDF以进行reset；若路径不存在，则尝试向/sys/bus/pci/devices/"PCI\_BDF"/reset路径写入1以进行reset；若路径仍不存在，则输出“The kernel doesn't support reset from sysfs for PCI device”提示。

备注：此处未找到do\_flr对应的文件节点，需确认其实现方式。

##### 1176-1183 函数 libxl\_\_device\_pci\_setdefault

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pci, bool hotplug

输出：int

描述：若pci设备的rdm（Raw Device Mapping）策略为LIBXL\_RDM\_RESERVE\_POLICY\_INVALID（-1），则将其改为LIBXL\_RDM\_RESERVE\_POLICY\_STRICT（0）。（此外，针对LIBXL\_RDM\_RESERVE\_POLICY\_RELAXED=1这个选项将不会进行特殊处理）

注释：

----在默认情况下，我们想强制保留特定于设备的rdm

----We'd like to force reserve rdm specific to a device by default.

##### 1185-1194 函数 libxl\_device\_pci\_add

输入：libxl\_ctx \*ctx, uint32\_t domid, libxl\_device\_pci \*pcidev, const libxl\_asyncop\_how \*ao\_how

输出：int

描述：先建立一个用于回调控制的ao，然后执行pci的添加流程（[1213-1310行](#_1213-1310_函数_libxl__device_pci_add)），最后检查ao的运行状态。

备注：此处涉及异步操作的[ao](#_<---关于_libxl__ao_结构体--->)相关内容。

##### 1196-1211 函数 libxl\_pcidev\_assignable

输入：libxl\_ctx \*ctx, libxl\_device\_pci \*pcidev

输出：int

描述：在所有已准备好直通的pci设备（由[399-446行](#_399-446_函数_libxl_device_pci_assigna)获得）中寻找输入的pci设备，若找到则返回1，未找到返回0。

##### 1213-1310 函数 libxl\_\_device\_pci\_add

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int starting

输出：int

描述：执行pci设备的直通、检测工作。先针对hvm模式的虚机测试xc的pci设备注册状态，若不可注册则提示无IOMMU或已被注册给其他虚机；然后调整pci设备的rdm策略状态（[1176-1183行](#_1176-1183_函数_libxl__device_pci_setd)）；再然后验证pci设备的seize参数（决定该pci设备是否自动进行assignable）并检查pci设备是否做好直通准备（[640-666行](#_640-666_函数_pciback_dev_is_assigned)，通过检测pciback下是否有该pci设备的link实现），以此判断是否需要在此处执行pci设备的直通准备工作（[753-828行](#_753-828_函数_libxl__device_pci_assign)）；然后验证此时pci设备是否已做好直通准备（[1196-1211行](#_1196-1211_函数_libxl_pcidev_assignabl)，通过遍历所有已做好直通准备的设备来做出这一判断，这里会排除已经标注为直通给其他虚拟机的设备），若未准备好则返回错误信息；再然后拉取所有已直通的设备列表及设备总数（[308-350行](#_308-350_函数_get_all_assigned_devices)），并在其中寻找当前pci设备（以验证该设备是否已被分配，尤其是当前虚机，可能会与上一步产生冗余）；然后对当前设备执行reset操作（[1139-1174行](#_1139-1174_函数_libxl__device_pci_rese)），并验证关于设备模型存根域（stubdom）相关的配置并执行相应操作（如果需要，会将该设备添加到对应的存根域中（使用[1010-1137行](#_1010-1137_函数_do_pci_add)实现）。存根域总是在运行的，即使在创建的时候）；再然后将准备直通后的设备号（vdevfn）后三位（二进制下）清零并存档（通过& ~7U实现），然后验证vdevfn小于等于7（否则报错）；然后检查该设备的功能掩码是否为LIBXL\_PCI\_FUNC\_ALL（全1）（若是，则检查该设备的所有功能是否都准备好被直通（[911-953行](#_911-953_函数_pci_multifunction_check)），若有未做好直通准备的设备则报错，无误的话记录下该设备的所有功能的掩码；否则将设备的实际功能号记录到功能掩码（局部临时变量pfunc\_mask）的对应位中）；最后依次判断pfunc\_mask记录的每一位，若最开始的设备的功能掩码为LIBXL\_PCI\_FUNC\_ALL，则添加该设备的所有功能到当前虚机，否则只找到当前设备并将该设备直通为功能号为0的pci设备到当前虚机（设备号仍旧可以指定）（执行添加pci设备的函数仍旧是[1010-1137行](#_1010-1137_函数_do_pci_add)）。

##### 1312-1340 函数 libxl\_\_add\_pcidevs

输入：libxl\_\_egc \*egc, libxl\_\_ao \*ao, uint32\_t domid, libxl\_domain\_config \*d\_config, libxl\_\_multidev \*multidev

输出：void

描述：将d\_config中的pci设备全部直通到当前虚机中，该函数为[libxl\_\_pcidev\_devtype](#_<---关于_libxl_device_type_结构体--->)（dt）中链接的.add操作（具体调用位置来自tools/libxl/libxl\_create.c的1531行，需注意向上链接过程中存在一些回调操作）。先建立相关的ao设备（以便于异步加速运行）；再将d\_config中的全部pci设备进行直通操作（使用[1213-1310行](#_1213-1310_函数_libxl__device_pci_add)进行实现）；最后验证当直通的设备数大于0（进行过直通操作）时，将相关参数记录进xs中。

##### 1342-1376 函数 qemu\_pci\_remove\_xenstore

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int force

输出：int

描述：通过qemu的接口从虚机中移除pci设备。只有当force为0且vdevfn的fun为0时才执行操作。

注释：

--只有当输入的设备模型功能号为0时才一次性的自动移除所有设备。

--是否会发生错误取决于guest操作系统是否对SCI有反应。如果它不能按时响应那么就要考虑强制移除设备。

##### 1378-1379 函数定义 libxl\_\_device\_pci\_remove\_common

实现部分链接[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)

##### 1381-1509 函数 do\_pci\_remove

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int force

输出：int

描述：先获取当前虚机上所有直通过来的pci设备（[1611-1636行](#_1611-1636_函数_libxl_device_pci_list)，若不存在则报错），然后若能在其中找到输入指定的pci设备（[352-370行](#_352-370_函数_is_pcidev_in_array)，不能就报错），则根据虚机的不同类型进行处理：针对HVM虚机，根据正在运行的dm\_version（qemu\_xen\_traditional或qemu\_xen）类型的不同，决定将remove操作交给当前函数中的qemu\_pci\_remove\_xenstore（[1342-1376行](#_1342-1376_函数_qemu_pci_remove_xensto)，force会被原封不动的传入）来执行，或交给libxl\_\_qmp\_pci\_del（然后转交给qemu）来执行；否则则应针对PV虚机（若仍不是，就报错），此时交给xc直接向/sys/bus/pci/devices/"PCI\_BDF"/路径进行操作，将其内存空间映射回来、中断信号恢复响应。然后，若恢复回来的设备直通后的功能号为0，则对其进行reset（[1139-1174行](#_1139-1174_函数_libxl__device_pci_rese)）；若存根域为0，则通过xc取消注册设备；重新识别存根域后，若不是0了，则将该设备从这一存根域移除（存在与[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)的递归循环调用，但每次的域id应该不一样）。最后，从xs中移除这个设备的记录（[192-306行](#_192-306_函数_libxl__device_pci_remove)）。

注释：设备仍旧有正在直通的功能时，不可以同时reset多个。

##### 1511-1544 函数 libxl\_\_device\_pci\_remove\_common

输入：libxl\_\_gc \*gc, uint32\_t domid, libxl\_device\_pci \*pcidev, int force

输出：int

描述：检查该设备的功能掩码是否为LIBXL\_PCI\_FUNC\_ALL（全1）（若是，则检查该设备的所有功能是否都准备好被直通（[911-953行](#_911-953_函数_pci_multifunction_check)，做好直通准备也意味着可以被移除），若有未做好直通准备的设备则报错，无误的话记录下该设备的所有功能的掩码；否则将设备的实际功能号记录到功能掩码（局部临时变量pfunc\_mask）的对应位中），然后根据之前的结果，来选择移除所有的功能，还是只找到并移除唯一指定的pci设备（具体的移除操作在[1381-1509行](#_1381-1509_函数_do_pci_remove)）。

备注：与[1381-1509行](#_1381-1509_函数_do_pci_remove)存在循环递归关系，可能和多重分配有关（同一个设备被不断的分配给下属的虚拟机）

##### 1546-1558 函数 libxl\_device\_pci\_remove

输入：libxl\_ctx \*ctx, uint32\_t domid, libxl\_device\_pci \*pcidev, const libxl\_asyncop\_how \*ao\_how

输出：int

描述：函数libxl\_\_device\_pci\_remove\_common（[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)）的封装，使用非强制模式，额外包含了异步控制结构的内容。

##### 1560-1571 函数 libxl\_device\_pci\_destroy

输入：libxl\_ctx \*ctx, uint32\_t domid, libxl\_device\_pci \*pcidev, const libxl\_asyncop\_how \*ao\_how

输出：int

描述：函数libxl\_\_device\_pci\_remove\_common（[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)）的封装，使用强制模式，额外包含了异步控制结构的内容。

##### 1573-1609 函数 libxl\_\_device\_pci\_from\_xs\_be

输入：libxl\_\_gc \*gc, const char \*be\_path, libxl\_device\_pci \*pci, int nr

输出：void

描述：从xs的后端路径中恢复pci的参数信息到输入的pci设备中。

##### 1611-1636 函数 libxl\_device\_pci\_list

输入：libxl\_ctx \*ctx, uint32\_t domid, int \*num

输出：libxl\_device\_pci \*

描述：从xs中读取当前虚机上所有（被直通过来）的pci设备。返回设备总数以及设备列表（数组）。

##### 1638-1659 函数 libxl\_\_device\_pci\_destroy\_all

输入：libxl\_\_gc \*gc, uint32\_t domid

输出：int

描述：remove当前dom下的所有pci设备。通过[1611-1636行](#_1611-1636_函数_libxl_device_pci_list)获得该虚拟机中的所有pci设备，具体的移除操作在[1511-1544行](#_1511-1544_函数_libxl__device_pci_remo)。

注释：

----在HVM模式下应当使用强制模式，因为在运行到这个函数的时候，会因为客户机的内核已经关掉了这个设备而使qemu无法响应SCI中断。

##### 1661-1704 函数 libxl\_\_grant\_vga\_iomem\_permission

输入：libxl\_\_gc \*gc, const uint32\_t domid, libxl\_domain\_config \*const d\_config

输出：int

描述：如果未开启gfx\_passthru选项，则直接返回0。然后会在d\_config中找到Class Code为0x030000（代表VGA类）的设备，然后通过xc向当前虚机对应的存根域传递vga的iomem许可，再向当前域传递vga的iomem许可，找到第一个VGA并完成流程后就会退出并返回0。

备注：包含了地址分配，且当前是固定值。

##### 1706-1710 函数 libxl\_device\_pci\_compare

输入：libxl\_device\_pci \*d1, libxl\_device\_pci \*d2

输出：int

描述：比较两个pci设备的BDF，全相同则返回1，否则返回0。（实际为一个连续等于、与操作的宏的封装）

##### 1712-1712 宏定义

#define libxl\_\_device\_pci\_update\_devid NULL

##### 1714-1714 函数声明

DEFINE\_DEVICE\_TYPE\_STRUCT\_X(pcidev, pci, PCI);

（参考112-190的结构体解说）

#### 其他文件中相关函数与宏

##### <--- 头文件内--->

----路径：tools/libxl/libxl\_osdeps.h

----25-63 生效的设定项：

----#define SYSFS\_USB\_DEV "/sys/bus/usb/devices"

----#define SYSFS\_USBBACK\_DRIVER "/sys/bus/usb/drivers/usbback"

----#define SYSFS\_PCI\_DEV "/sys/bus/pci/devices"

----#define SYSFS\_PCIBACK\_DRIVER "/sys/bus/pci/drivers/pciback"

----#define NETBACK\_NIC\_NAME "vif%u.%d"

----#include <sys/sysmacros.h>

----#include <pty.h>

----#include <uuid/uuid.h>

----65-77 验证与USB、PCI相关的宏定义，提示错误信息

----79-84 定义了两种打印模式：asprintf与vasprintf

----86-120 定义了关于大小端数据模式的分别处理

<---头文件结束--->

##### <---关于 libxl\_device\_pci 结构体--->

----路径：tools/libxl/\_libxl\_types.h：886

----描述：定义关于pci设备的相关参数。

----typedef struct libxl\_device\_pci {

---- uint8\_t func; //功能号

---- uint8\_t dev; //设备号

---- uint8\_t bus; //总线号

---- int domain; //终端号

---- uint32\_t vdevfn; //指定直通后的设备号

---- uint32\_t vfunc\_mask; //包含了当前设备的所有功能号

---- bool msitranslate; //MSI与INITx中断转换（强制使用MSI中断）

---- bool power\_mgmt; //电源管理（VM对D0-D3hot的控制）

---- bool permissive; //允许对设备的完全控制（可能导致安全性与稳定性问题）

---- bool seize; //启用自动分配pci设备（自动进行pci-assignable-add）

---- libxl\_rdm\_reserve\_policy rdm\_policy; //关于保留设备内存的配置（默认为空，有可能导致客户机崩溃）

----} libxl\_device\_pci;

<---结构体定义结束--->

##### <---关于 libxl\_\_gc 结构体--->

----路径：tools/libxl/libxl\_internal.h：432

----描述：用于设计垃圾回收管理相关结构。

----struct libxl\_\_gc {

---- /\* mini-GC \*/

---- int alloc\_maxsize; /\* -1 means this is the dummy non-gc gc \*/

---- void \*\*alloc\_ptrs;

---- libxl\_ctx \*owner;

----};

<---结构体定义结束--->

##### <---关于 flexarray\_t 结构体--->

----路径：tools/libxl/flexarray.h：21

----描述：用于设计垃圾回收管理相关结构。

----typedef struct flexarray {

---- int size;

---- int autogrow;

---- unsigned int count;

---- void \*\*data; /\* array of pointer \*/

---- struct libxl\_\_gc \*gc;

----} flexarray\_t;

<---结构体定义结束--->

##### <---关于 xenbus\_state （XenbusState）枚举类型--->

----路径：tools/include/xen/io/xenbus.h：36

----描述：用于定义Xen总线设备的运行状态。在该文件夹下的其他头文件中包含其各自初始化流程示意图。

----enum xenbus\_state {

---- XenbusStateUnknown = 0,

---- XenbusStateInitialising = 1,

---- XenbusStateInitWait = 2, // InitWait: Finished early initialisation but waiting for information from the peer or hotplug scripts.

---- XenbusStateInitialised = 3, // Initialised: Waiting for a connection from the peer.

---- XenbusStateConnected = 4,

---- XenbusStateClosing = 5, // Closing: The device is being closed due to an error or an unplug event.

---- XenbusStateClosed = 6,

---- XenbusStateReconfiguring = 7, // Reconfiguring: The device is being reconfigured.

---- XenbusStateReconfigured = 8

----};

----typedef enum xenbus\_state XenbusState;

<---枚举类型定义结束--->

##### <---关于 libxl\_device 结构体--->

----路径：tools/libxl/libxl\_internal.h：512

----描述：定义关于设备的相关参数，包括其后端类型与实际类型。

----/\* libxl\_\_device is a transparent structure that doesn't contain private fields or external memory references, and as such can be copied by assignment. \*/

----typedef struct {

---- uint32\_t backend\_devid;

---- uint32\_t backend\_domid;

---- uint32\_t devid;

---- uint32\_t domid;

---- libxl\_\_device\_kind backend\_kind;

---- libxl\_\_device\_kind kind;

----} libxl\_\_device;

<---结构体定义结束--->

##### <---关于 libxl\_\_device\_kind 枚举类型--->

----路径：tools/libxl/\_libxl\_types\_internal.h：24

----描述：用于定义设备类型。

----typedef enum libxl\_\_device\_kind {

---- LIBXL\_\_DEVICE\_KIND\_NONE = 0,

---- LIBXL\_\_DEVICE\_KIND\_VIF = 1,

---- LIBXL\_\_DEVICE\_KIND\_VBD = 2,

---- LIBXL\_\_DEVICE\_KIND\_QDISK = 3,

---- LIBXL\_\_DEVICE\_KIND\_PCI = 4,

---- LIBXL\_\_DEVICE\_KIND\_VFB = 5,

---- LIBXL\_\_DEVICE\_KIND\_VKBD = 6,

---- LIBXL\_\_DEVICE\_KIND\_CONSOLE = 7,

---- LIBXL\_\_DEVICE\_KIND\_VTPM = 8,

---- LIBXL\_\_DEVICE\_KIND\_VUSB = 9,

---- LIBXL\_\_DEVICE\_KIND\_QUSB = 10,

---- LIBXL\_\_DEVICE\_KIND\_9PFS = 11,

---- LIBXL\_\_DEVICE\_KIND\_VDISPL = 12,

---- LIBXL\_\_DEVICE\_KIND\_VUART = 13,

---- LIBXL\_\_DEVICE\_KIND\_PVCALLS = 14,

----} libxl\_\_device\_kind;

<---枚举类型定义结束--->

##### libxl\_\_xs\_read：（tools/libxl/libxl\_xshelp.c）

向gc->owner->xsh->fd（关系到gc对应的domain编号，此处为0？）写入字符串（向xenstore传递消息），然后将返回的字符串存入gc进行管理且返回字符串。

##### libxl\_\_xs\_writev：（tools/libxl/libxl\_xshelp.c）

向gc->owner->xsh->fd写入字符串（向xenstore传递消息），只返回状态值。

##### atoi：（<stdlib.h>）

将输入的字符串转换为长整数，若未执行有效转换则返回0。

##### device\_add\_domain\_config：（tools/libxl/libxl\_device.c）

根据输入pcidev设备的func、bus、dev尝试在d\_config中找到对应的目标，若未找到则为d\_config中对应数组声明新的空间并将临时变量item指向这一空间，然后初始化item并将pci相关信息复制进去；若找到，则弃用item。

##### <---关于 libxl\_domain\_config 结构体--->

----路径：tools/libxl/\_libxl\_types.h：1047

----描述：源自cfg文件设定值，用于配置虚机的启动信息。

----typedef struct libxl\_domain\_config {

---- libxl\_domain\_create\_info c\_info;

---- libxl\_domain\_build\_info b\_info;

---- int num\_disks;

---- libxl\_device\_disk \* disks;

---- int num\_nics;

---- libxl\_device\_nic \* nics;

---- int num\_pcidevs;

---- libxl\_device\_pci \* pcidevs;

---- int num\_rdms;

---- libxl\_device\_rdm \* rdms;

---- int num\_dtdevs;

---- libxl\_device\_dtdev \* dtdevs;

---- int num\_vfbs;

---- libxl\_device\_vfb \* vfbs;

---- int num\_vkbs;

---- libxl\_device\_vkb \* vkbs;

---- int num\_vtpms;

---- libxl\_device\_vtpm \* vtpms;

---- int num\_p9s;

---- libxl\_device\_p9 \* p9s;

---- int num\_pvcallsifs;

---- libxl\_device\_pvcallsif \* pvcallsifs;

---- int num\_vdispls;

---- libxl\_device\_vdispl \* vdispls;

---- int num\_channels;

---- libxl\_device\_channel \* channels;

---- int num\_usbctrls;

---- libxl\_device\_usbctrl \* usbctrls;

---- int num\_usbdevs;

---- libxl\_device\_usbdev \* usbdevs;

---- libxl\_action\_on\_shutdown on\_poweroff;

---- libxl\_action\_on\_shutdown on\_reboot;

---- libxl\_action\_on\_shutdown on\_watchdog;

---- libxl\_action\_on\_shutdown on\_crash;

---- libxl\_action\_on\_shutdown on\_soft\_reset;

----} libxl\_domain\_config;

<---结构体定义结束--->

/\*

----信息从config文件到config结构体的赋值位置在xl\_parse.c：854中的parse\_config\_data函数中。

----libxl\_device\_type结构体中存有上述一些值的偏移量。

\*/

##### <---关于 libxl\_device\_type 结构体--->

----路径：tools/libxl/\_libxl\_internal.h：3581

----描述：常见被命名为“dt”，会调用该结构体指向的函数指针进行初始化等操作。

----struct libxl\_device\_type {

---- libxl\_\_device\_kind type;

---- int skip\_attach; /\* Skip entry in domcreate\_attach\_devices() if 1 \*/

---- int ptr\_offset; /\* Offset of device array ptr in libxl\_domain\_config \*/

---- int num\_offset; /\* Offset of # of devices in libxl\_domain\_config \*/

---- int dev\_elem\_size; /\* Size of one device element in array \*/

---- device\_add\_fn\_t add;

---- device\_set\_default\_fn\_t set\_default;

---- device\_to\_device\_fn\_t to\_device;

---- device\_init\_fn\_t init;

---- device\_copy\_fn\_t copy;

---- device\_dispose\_fn\_t dispose;

---- device\_compare\_fn\_t compare;

---- device\_merge\_fn\_t merge;

---- device\_dm\_needed\_fn\_t dm\_needed;

---- device\_update\_config\_fn\_t update\_config;

---- device\_update\_devid\_fn\_t update\_devid;

---- device\_from\_xenstore\_fn\_t from\_xenstore;

---- device\_set\_xenstore\_config\_fn\_t set\_xenstore\_config;

----};

----结构体初始化：tools/libxl/\_libxl\_internal.h：3602

----#define DEFINE\_DEVICE\_TYPE\_STRUCT\_X(name, sname, kind, ...) \

---- const struct libxl\_device\_type libxl\_\_ ## name ## \_devtype = { \

---- .type = LIBXL\_\_DEVICE\_KIND\_ ## kind, \

---- .ptr\_offset = offsetof(libxl\_domain\_config, name ## s), \

---- .num\_offset = offsetof(libxl\_domain\_config, num\_ ## name ## s), \

---- .dev\_elem\_size = sizeof(libxl\_device\_ ## sname), \

---- .add = libxl\_\_add\_ ## name ## s, \

---- .set\_default = (device\_set\_default\_fn\_t) \

---- libxl\_\_device\_ ## sname ## \_setdefault, \

---- .to\_device = (device\_to\_device\_fn\_t)libxl\_\_device\_from\_ ## name, \

---- .init = (device\_init\_fn\_t)libxl\_device\_ ## sname ## \_init, \

---- .copy = (device\_copy\_fn\_t)libxl\_device\_ ## sname ## \_copy, \

---- .dispose = (device\_dispose\_fn\_t) \

---- libxl\_device\_ ## sname ## \_dispose, \

---- .compare = (device\_compare\_fn\_t) \

---- libxl\_device\_ ## sname ## \_compare, \

---- .update\_devid = (device\_update\_devid\_fn\_t) \

---- libxl\_\_device\_ ## sname ## \_update\_devid, \

---- \_\_VA\_ARGS\_\_ \

---- }

----#define DEFINE\_DEVICE\_TYPE\_STRUCT(name, kind, ...) \

---- DEFINE\_DEVICE\_TYPE\_STRUCT\_X(name, name, kind, \_\_VA\_ARGS\_\_)

----全局变量：tools/libxl/\_libxl\_internal.h：3645

----extern const struct libxl\_device\_type libxl\_\_vfb\_devtype;

----extern const struct libxl\_device\_type libxl\_\_vkb\_devtype;

----extern const struct libxl\_device\_type libxl\_\_disk\_devtype;

----extern const struct libxl\_device\_type libxl\_\_nic\_devtype;

----extern const struct libxl\_device\_type libxl\_\_vtpm\_devtype;

----extern const struct libxl\_device\_type libxl\_\_usbctrl\_devtype;

----extern const struct libxl\_device\_type libxl\_\_usbdev\_devtype;

----extern const struct libxl\_device\_type libxl\_\_pcidev\_devtype;

----extern const struct libxl\_device\_type libxl\_\_vdispl\_devtype;

----extern const struct libxl\_device\_type libxl\_\_p9\_devtype;

----extern const struct libxl\_device\_type libxl\_\_pvcallsif\_devtype;

----

----extern const struct libxl\_device\_type \*device\_type\_tbl[];

<---结构体定义结束--->

/\*

----dt提供的是一系列方式，表示一组方法，相关函数很善于通过使用指针、偏移量来获取相关值。

----一些针对于pci的dt操作：const struct libxl\_device\_type libxl\_\_pcidev\_devtype

----.type = LIBXL\_\_DEVICE\_KIND\_PCI,

----.ptr\_offset = offsetof(libxl\_domain\_config, pcidevs),

----对应libxl\_domain\_config中\* pcidevs的相对偏移量

----.num\_offset = offsetof(libxl\_domain\_config, num\_pcidevs),

----对应libxl\_domain\_config中num\_pcidevs的相对偏移量

----.dev\_elem\_size = sizeof(libxl\_device\_pci),

----[.add](#_1312-1340_函数_libxl__add_pcidevs) = libxl\_\_add\_pcidevs,

----.set\_default = (device\_set\_default\_fn\_t) libxl\_\_device\_pci\_setdefault,

----.to\_device = (device\_to\_device\_fn\_t)libxl\_\_device\_from\_pci,

----.init = (device\_init\_fn\_t)libxl\_device\_pci\_init,

----.copy = (device\_copy\_fn\_t)libxl\_device\_pci\_copy,

----结构体内容依次复制，（右侧复制到左侧）

----.dispose = (device\_dispose\_fn\_t) libxl\_device\_pci\_dispose,

----指针内容全部清零

----[.compare](#_1706-1710_函数_libxl_device_pci_compa) = (device\_compare\_fn\_t) libxl\_device\_pci\_compare,

----比较func、bus、dev三个参数（BDF），全相同则返回真

----.update\_devid = (device\_update\_devid\_fn\_t) libxl\_\_device\_pci\_update\_devid,

\*/

##### <---关于 libxl\_domain\_type 枚举类型--->

----路径：tools/libxl/\_libxl\_types.h：44

----描述：用于定义虚机类型。

----typedef enum libxl\_domain\_type {

---- LIBXL\_DOMAIN\_TYPE\_INVALID = -1,

---- LIBXL\_DOMAIN\_TYPE\_HVM = 1,

---- LIBXL\_DOMAIN\_TYPE\_PV = 2,

---- LIBXL\_DOMAIN\_TYPE\_PVH = 3,

----} libxl\_domain\_type;

<---枚举类型定义结束--->

##### <---关于 libxl\_asyncop\_how 结构体--->

----路径：tools/libxl/libxl.h：1287

----描述：用于配置并发的异步任务操作。

----typedef struct {

---- void (\*callback)(libxl\_ctx \*ctx, int rc, void \*for\_callback);

---- union {

---- libxl\_ev\_user for\_event; /\* used if callback==NULL \*/

---- void \*for\_callback; /\* passed to callback \*/

---- } u;

----} libxl\_asyncop\_how;

----注释：

--有些libxl的操作可能会花费很长的时间。下列函数提供了一种可以被用于控制他们的并发性的变量：

----libxl\_asyncop\_how \*ao\_how

--如果ao\_how==NULL，这个函数将会是同步的；如果ao\_how!=NULL，函数将设置操作应该如何进行，如果设置成功，则返回0。在这种情况下，返回值为0并不一定意味着操作是成功的；它只是意味着操作已经被成功启动。它将在稍后完成，那时的结果也许会是错误的。

--如果ao\_how->callback!=NULL，当操作完成时将调用callback指向的函数。此处适用与libxl\_event\_hooks相同的规则，包括重新运行规则和“灾难”的可能性，但libxl将调用ao\_how->callback而不是libxl\_event\_hooks.event\_occurs。（参考libxl\_event.h）

--如果ao\_how->callback==NULL，将会生成一个可以从libxl\_event\_wait或libxl\_event\_check中获得的libxl\_event，这个event将会具有OPERATION\_COMPLETE类型的标记（这一标记只会在这里使用）。

--请注意，这里可能会产生异步操作，它将导致回调在其发起函数调用期间完成。在上述情形中，当操作处于“in progress”状态时，启动函数就已经返回0，这样就会导致即使在返回时操作已经完成，回调也已经发生了。

--应用程序必须设置和使用ao\_how->for\_event（它将被复制到libxl\_event.for\_user中）或ao\_how->for\_callback（它将被传递到callback中）来确定哪个操作将会被完成，它必须检查rc值是否有错误。

--\*ao\_how不需要在启动函数返回后保持有效。除非另有规定，所有其他参数必须在异步操作的生存期内保持有效。

--这些回调可能发生在应用程序调用libxl的任何线程上。

<---结构体定义结束--->

##### <---关于 libxl\_\_ao 结构体--->

----路径：tools/libxl/libxl\_internal.h：565

----描述：用于配置并发的多线程任务操作。

----struct libxl\_\_ao {

---- uint32\_t magic;

---- unsigned constructing:1, in\_initiator:1, complete:1, notified:1,

---- aborting:1;

---- int manip\_refcnt;

---- libxl\_\_ao \*nested\_root;

---- int nested\_progeny;

---- int progress\_reports\_outstanding;

---- int rc;

---- LIBXL\_LIST\_HEAD(, libxl\_\_ao\_abortable) abortables;

---- LIBXL\_LIST\_ENTRY(libxl\_\_ao) inprogress\_entry;

---- libxl\_\_gc gc;

---- libxl\_asyncop\_how how;

---- libxl\_\_poller \*poller;

---- uint32\_t domid;

---- LIBXL\_TAILQ\_ENTRY(libxl\_\_ao) entry\_for\_callback;

----};

----（通常由[ao\_how](#_<---关于_libxl_asyncop_how_结构体--->)进行配置、处理）

----注释：

--只有持有ctx锁，才能访问ao及其gc。

--特殊例外：如果ao已被添加到egc->aos\_for\_callback中，则拥有这个egc的线程可以从该列表中移除ao并在不持有锁的情况下进行回调。

--相应的限制：一个ao一次只可以添加到一个egc->aos\_for\_callback中；rc和how必须已经被设置，并且在之后不可以修改。（由于ao只在libxl\_\_ao\_complete中排队等待回调，因此很容易且明显地可以满足这一限制。）

<---结构体定义结束--->

#### 一些重要的功能说明

参考链接：

<http://xenbits.xen.org/docs/unstable/man/xl.cfg.5.html>

##### device\_model\_version="DEVICE-MODEL"

Selects which variant of the device-model should be used for this guest.

Valid values are:

qemu-xen

Use the device-model merged into the upstream QEMU project. This device-model is the default for Linux dom0.

qemu-xen-traditional

Use the device-model based upon the historical Xen fork of QEMU. This device-model is still the default for NetBSD dom0.

It is recommended to accept the default value for new guests. If you have existing guests then, depending on the nature of the guest Operating System, you may wish to force them to use the device model which they were installed with.

## 附件B：tools/libxl/flexarray相关

### flexarray.h 共55行

进行了关于

/\*

\* Copyright (C) 2009 Citrix Ltd.

\* Author Vincent Hanquez <vincent.hanquez@eu.citrix.com>

\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU Lesser General Public License as published

\* by the Free Software Foundation; version 2.1 only. with the special

\* exception on linking described in file LICENSE.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU Lesser General Public License for more details.

\*/

#ifndef FLEXARRAY\_H

#define FLEXARRAY\_H

struct libxl\_\_gc;

typedef struct flexarray {

int size; //当前总空间（最大空间）

int autogrow; //是否为自增长模式

unsigned int count; //当前已占用空间（初始化为0）

void \*\*data; /\* array of pointer \*/

struct libxl\_\_gc \*gc;

} flexarray\_t;

/\*

\* NOGC can be used with flexarrays, but flexarray\_free will need to be called

\* to free the struct. The content of the flexarray will not be freed through

\* flexarray\_free.

NOGC模式下仍旧可以使用flexarray，但是需要调用flexarray\_free函数以释放这一结构体。尽管flexarray的内容不会被flexarray\_free函数释放。

\*/

\_hidden flexarray\_t \*flexarray\_make(struct libxl\_\_gc \*gc\_opt,

int size, int autogrow);

\_hidden void flexarray\_free(flexarray\_t \*array);

\_hidden void flexarray\_grow(flexarray\_t \*array, int extents);

\_hidden int flexarray\_set(flexarray\_t \*array, unsigned int index, void \*ptr);

\_hidden int flexarray\_append(flexarray\_t \*array, void \*ptr);

\_hidden int flexarray\_append\_pair(flexarray\_t \*array, void \*ptr1, void \*ptr2);

\_hidden int flexarray\_vappend(flexarray\_t \*array, ...);

\_hidden int flexarray\_get(flexarray\_t \*array, int index, void \*\*ptr);

\_hidden void \*\*flexarray\_contents(flexarray\_t \*array);

/\*

其中，在tools/libxl/libxl\_internal.h：76中，对于\_hidden关键字有如下定义，

#if \_\_GNUC\_\_ > 3 || (\_\_GNUC\_\_ == 3 && \_\_GNUC\_MINOR\_\_ >= 1)

#define \_hidden \_\_attribute\_\_((visibility("hidden")))

#define \_protected \_\_attribute\_\_((visibility("protected")))

#else

#define \_hidden

#define \_protected

#endif

即，使用\_hidden参数可以通过将该函数设为外部不可见，从而避免.so中的同名符号冲突。

\*/

#endif

/\*

\* Local variables:

\* mode: C

\* c-basic-offset: 4

\* indent-tabs-mode: nil

\* End:

\*/